SIMULATED ANNEALING ON NP-COMPLETE PROBLEMS

Russell W. Howell
Department of Mathematics and Computer Science
Westmont College
Santa Barbara, California

INTRODUCTION:

The *Random House Dictionary* defines *anneal* as "...to free (glass, metals, etc.) from internal stress by heating and gradually cooling." In the physical world, impurities are *annealed out* of a substance by heating it to a high temperature. As it is cooled, its molecular structure gradually settles into a stable "low-energy" configuration. It is important to cool the substance slowly, especially at lower temperatures, so that impurities do not get "frozen" into place. With careful cooling, a low-energy state can eventually be reached. This state may not be the one with the *lowest* potential energy, where the spins of the electrons are uniformly aligned, but it is usually one where there are very few domains-each one containing electrons with approximately aligned spins.

There is a surprising analogy between the process described above and an algorithm that has proven to be successful in large classes of combinatorial optimization problems. The algorithm was formally introduced in its present form in 1983 by Kirkpatrick, Gelatt and Vecchi [5], although slightly different versions were used as early as 1953 by Metropolis et al. [9]. Because the algorithm searches for "optimized states" of certain combinatorial problems, and because it is guided by a control parameter (temperature) which is initially large and gradually becomes small, the algorithm has been given the name *simulated annealing*.

The idea of simulated annealing is simple. We begin with a certain structure, such as a list of cities which a salesperson is to visit, and associate with that structure a *cost*, such as the distance involved in traveling to all the cities in the order specified by the list. We wish to obtain a configuration of this structure which has minimum cost. The algorithm proceeds as follows:

1. **Initialize:**
   a) The system configuration.
   b) The *cost* of the system configuration.
   c) The current control parameter, *temperature*.
2. **Repeat:**
   a) Randomly alter the current system configuration, obtaining a (potentially) new configuration.
   b) Evaluate the *new cost* of this "candidate" system configuration.
   c) **If** *new cost* $<$ *old cost* **then**
      accept the candidate system configuration as the current configuration.
      else {*new cost* $\geq$ *old cost*}
      begin
      Select a random number (between 0 and 1).
      If the random number is less than a temperature-dependent quantity **then**
      accept the new configuration as the current configuration.
      else
      keep the old configuration as the current configuration.
      end

d) Lower *temperature*.
Until who knows when!

The "temperature-dependent" quantity referred to above is actually a Boltzman probability and is a function of the old cost of the structure, the new cost, and the temperature. If r is the random number selected, the new configuration is accepted if and only if

$$r < e^{-\frac{(\text{new cost} - \text{old cost})}{\text{temperature}}}.$$

The intuitive idea behind allowing "uphill-steps" is that the algorithm will not get "trapped" in a local minimum, as might be the case if a deterministic descent technique were used instead. Initially the temperature is high, so most uphill steps are accepted. Perhaps at high temperatures the global properties of the structure to be optimized are sorted out, as is the case with the corresponding physical process. As the temperature is lowered, it becomes harder to accept cost increases, and it is perhaps here that the local structure properties are fine-tuned.

Other schemes for accepting or rejecting the new configuration have been suggested [10]. One obvious change to the algorithm which was used for all the examples in this paper is to keep track of the lowest cost obtained and update that variable (as well as saving the state of the structure giving rise to that cost) whenever a new lower cost is found. When the algorithm terminates, this "lowest cost" state is used as the final state.

The question of when to stop the algorithm is problematic. In practice it is halted when either the temperature has dropped *below* a certain critical cut-off value, or the number of iterations of step two above has *exceeded* a certain cut-off value. Besides the determination of stopping conditions, other questions which come to mind are:

1. Is there any reason to expect that the algorithm will *work*? That is, what is the probability that the algorithm will converge to the global minimum, or to some value close to it?

2. How fast, if at all, does the algorithm converge? What topological properties of the problem structure are relevant?

3. What effect, if any, does the temperature have on the convergence speed of the algorithm? In particular, *how* should the temperature be varied.

In this paper we shall discuss these and other questions while applying the algorithm to three classes of optimization problems, all of which are known to be NP-complete: The traveling salesman, graph optimal linear arrangement, and rectangle packing. We begin with a review of what computer scientists mean by the term "NP-complete."

NP-COMPLETE PROBLEMS:

Definitions:

1. Given the alphabet $\Sigma=\{0,1\}$, we say that a *problem* f: $\Sigma^* \rightarrow \Sigma$ belongs to the class P (for polynomial) iff there exists a Turing machine T which computes f in polynomial-time. (Note: this definition can easily be extended to functions g: $\Sigma^* \rightarrow \Sigma^*$.)

To understand the above, imagine a problem we wish to solve as being recast in the form of a question. (For example, in the case of the traveling salesman, imagine being given a list of cities and a positive integer d, and ask whether there exists an ordering of the cities such that the total length of the tour specified by this ordering is *less* than d.) Next, come up with a scheme to "encode" this problem with a sequence of 0's and 1's. The function f (which encapsulates the problem) takes the encoding and returns 0 or 1 according to whether the answer to the question is false or true. Likewise, think of a Turing machine as being an *algorithm* which we can systematically apply to answer our question. The number of steps required to complete the algorithm will, of course, depend on the *size* of the problem (e.g., how many cities we have to consider). Loosely speaking, to say that T computes f in polynomial-time means that the algorithm gives the same result as the function f, and that the number of steps taken by the algorithm grows no faster with the size of the problem than does some polynomial.

2. A problem f is said to belong to the class NP iff there exists a polynomial p and a problem g such that

a) g belongs to the class **P**, and

b) $f(x)$ iff $g(xy)$ for some $y \in \Sigma^{p(|x|)}$,

where xy is the concatenation of x and y, and $|x|$ denotes the size of x.

In the above, we should think of x as being an *instance* of the problem f (e.g., a list of cities and a positive integer d), and y as a *solution* to the problem (e.g., a final ordering of the cities). To be in the class NP, then, means that in some sense *checking* whether the problem has an acceptable solution is easy, because the verification takes no longer than polynomial-time (since $g \in \mathbf{P}$), but *finding* a solution is potentially difficult, because any algorithmic search for one *may* take exponentially long (since $y \in \Sigma^{p(|x|)}$). The notion of NP-complete is related strongly to the concept of polynomial-time reducibility, which is defined next.

3. Given two problems f and g, we say that f is polynomial-time reducible to g (notation $f \leq_p g$) iff there exists a function $\rho: \Sigma^* \rightarrow \Sigma^*$ such that

a) $\rho \in \mathbf{P}$, and

b) $f(x)$ iff $g(\rho(x))$.

Thus, if f is polynomial-time reducible to g, in some sense f is "not much more difficult to compute" than g, for given a problem instance x, we can simply apply our "subroutine" ρ to compute $\rho(x)$ easily (i.e., in polynomial-time), and then apply g to this result to get $f(x)$.

4. A problem f is said to be NP-complete iff

a) $f \in NP$ and

b) $g \in NP$ implies $g \leq_p f$.

In an informal sense, then, an NP-complete problem f is as hard to compute as any other problem in the class NP, because any other problem g in the class NP is "not much more difficult to compute" than the problem f.

Of course, $\mathbf{P} \subseteq$ NP, and one of the biggest open questions in computer science is whether or not the classes NP and **P** coincide. It can be shown that if any NP-complete

problem has a polynomial-time algorithm, then NP and P would be identical. The overwhelming consensus seems to be that the two classes are not the same. If this is true, then no problem in the NP-complete class can be solved with a polynomial-time algorithm. Partially because of this, simulated annealing is of tremendous interest. We shall now discuss its use in more detail.

THE TRAVELING SALESMAN:

Simulated annealing seems to work remarkably well when it is applied to the traveling salesman problem [5, 7], but there is a surprising difference between performances arising from topologies associated with different perturbation schemes (step 2a of the algorithm). In what follows we shall consider two methods for randomly altering the system configuration (i.e., the specified ordering of the cities). The first simply swaps at random a pair of cities from the current configuration list, while the second, suggested by Karp, additionally reverses the route between the two cities. Thus, if the original list were

$$(C_1,\ldots,C_{i-1},C_i,C_{i+1},\ldots,C_{j-1},C_j,C_{j+1},\ldots,C_N)$$

and cities C_i and C_j were swapped, the first scheme would result in the list

$$(C_1,\ldots,C_{i-1},C_j,C_{i+1},\ldots,C_{j-1},C_i,C_{j+1},\ldots,C_N),$$

while the second would produce

$$(C_1,\ldots,C_{i-1},C_j,C_{j-1},\ldots,C_{i+1},C_i,C_{j+1},\ldots,C_N).$$

Figure 1a gives the initial route for a tour of 64 cities grouped randomly at each corner of a 100×100 square. Figures 1b-d show the remarkable progress the annealing method makes on typical runs using the second swapping method. Note that a final ordering which is obviously at least *close* to the global minimum has been found in no more than 40,000 steps, the (arbitrary) stopping point chosen for this application.

Figure 2 shows a comparison after 50,000 iterations between a) the first perturbation scheme and b) the second. The difference is apparently no accident, as the results of a *t*-test given in figure 3 suggest.

Intuitively, route-reversal will tend to cause better reductions in the cost function if the pair selected for swapping are terminating points of a "cross pattern" as indicated in figure 4. Obviously, Karp's procedure relies heavily on the Euclidean structure of the problem. A structure where the cost in traveling from one point to another were *not* proportional to the distance between them would not necessarily benefit from this method. In addition, there may be structures where it is meaningless to speak of the distance between two points, yet the cost *is* a function of the ordering of these points. Graph optimal linear arrangement is one such example.

GRAPH OPTIMAL LINEAR ARRANGEMENT:

Consider n circuit elements (chips, pla's, etc.), and information indicating how many connections exist between any two of them. We can represent this with a labeled graph on n vertices, where the label for each edge is the number of connections between the corresponding elements represented by the vertices. If we were to draw this graph in a horizontal row, its *density* is defined to be the maximum sum of the labels on the edges passing between any two vertices (see figure 5). The graph optimal linear arrangement (GOLA) problem seeks a linear ordering of these n verticies with minimum density.

Nahar, Sahni, and Shargowitz have shown that simulated annealing may not perform very well on GOLA problem instances [10]. In fact, they claim that the criterion for accepting cost increases is computationally expensive, and performs no better than a decision to accept arbitrarily increase costs once out of every eighteen opportunities.

The reasons for this are not easy to determine. It may be that there are several equal local minima for this type of a problem, and that various methods are therefore effective in finding them. To help see if this is the case, we constructed a tree of 40 vertices, $\{v_1,\ldots,v_{40}\}$ with edges $\langle v_i, v_{i+1}\rangle$, $(i = 1,\ldots 39)$. Then the vertices were altered to present an initial configuration of

$$\{v_{40}, v_{38},\ldots, v_2, v_1, v_3,\ldots, v_{37}, v_{39}\}.$$

There are only two permutations of the orderings in this case which give rise to the obvious best cost of 1. The cost of the initial configuration is 39, as that is how many edges cross the $v_2 - v_1$ gap. Figure 6a shows the results of comparing the cost decreases at various stages. The acceptance criterion for annealing does indeed appear to work better than the arbitrary criterion proposed by Nahar et. al. Further, by allowing for multiple edges between the vertices in this instance we may be able to make the cost differences appear as large as we wished. Figure 6b shows a comparison of Nahar's method with that of simulated annealing for a traveling salesman problem instance.

Since both the traveling salesman and GOLA problems are NP-complete, either one can be polonomially-time reduced to the other. In an intuitive sense, then, one would expect that any randomized algorithm which works well on one type of problem would also work well on the other. Does annealing perform better on the traveling salesman than it does on GOLA? If so, this would be a strange phenomenon.

The problems investigated so far were both one-dimensional in the sense that the system configuration was specified by a linear ordering of certain components. Rectangle packing, by contrast, requires more than a single linear constraint.

RECTANGLE PACKING:

Given a collection of rectangles of varying sizes and another large rectangle, *rectangle packing* seeks to determine whether there is a way of packing the collection inside the large rectangle in such a way that the small rectangles are packed in a non-overlapping, mutually orthogonal manner. In the annealing version of this problem, we seek to pack the rectangles into a square with minimum perimeter, allowing any packed rectangle to rotate 90 degrees. This type of problem arises naturally in many areas (e.g. VLSI design).

As stated, this is a continuous problem since the small rectangles can be placed, in principle, anywhere inside the square. Fortunately, Jerrum [4] has shown a way to reduce this continuous problem to an equivalent one that is discrete.

Given a set S, the pair $\langle R,U\rangle$ of partial orders on S is said to be *complementary* iff every element of $[(S \times S) - I]$ is comparable in *exactly* one of the two orders. The key to Jerrum's method lies in the following results which he proves [4, pages 3-5]:

1. If L and M are any pair of linear orders on S, then $L \cap M$ and $L \cap M^{-1}$ are complementary. In fact, *all* complementary partial orders can be expressed in this way.

2. If P and Q are partial orders on S with every element of $S \times S$ in the symmetric closure of P or Q (or both), then there exists a complementary pair of partial orders $\langle P_o, Q_o \rangle$ with $P_o \subseteq P$, $Q_o \subseteq Q$.

With these results, solving the rectangle packing problem is rather straightforward. We begin with an ordered set A of n rectangles and generate two random permutations L and M of the set $S = \{1,\ldots,n\}$ representing two linear orders on S. By what was said above, the pair $\langle R,U \rangle = \langle L \cap M, L \cap M^{-1} \rangle$ are complementary. In terms of placing the rectangles, R can be thought of as representing a *right* partial order, and U an *upper* partial order. Because L appears as itself in both R and U, we may choose from L the *smallest* element with respect to R and U and place the rectangle corresponding with that element at position (0,0). By Jerrum's (rather surprising) second result, the placement of the remaining rectangles given by the linear order L may be *constrained* by forcing them as low or to the left as the size of the previously placed rectangles will allow, but ensuring that the placement is still consistent with the complementary pair $\langle R,U \rangle$.

Randomly altering our "system configuration" is now easy. All we need to do is perturb the L and/or M linear orders on S, recompute $\langle R,U \rangle$, and change the orientation of randomly selected rectangles by 90 degrees. To try out this perturbation scheme, we created an instance of 30 randomly sized rectangles, where the initial configuration was the placement of the rectangles end-to-end. Figure 7 shows the result of applying the annealing algorithm after 1000 and 50,000 iterations respectively. If the dimensions of the large rectangle were (x,y), the cost function used was

$$c(x,y) = 2 \times \max\{x-s, y-s\} + \min\{x-s, y-s\},$$

where s was the side of the square whose area equaled the sum of the areas of the small rectangles. This tended to force a packing into a square rather than a long thin rectangle.

The result of using annealing in this case is impressive. Have we just been lucky here (as with the traveling salesman), or are there theoretical reasons as to why the annealing algorithm should work?

THEORETICAL CONSIDERATIONS:

The usual perspective taken in order to develop theoretical results is to view annealing as a Markov chain on a finite state space X, with each possible configuration being given as an element $x_i \in X$, $i = 1,\ldots,|X|$.

PRELIMINARIES AND BASIC NOTATION:

The goal of annealing is to find a point $x_i \in X$ that minimizes a cost function $c: X \to R$. We shall write c_i for $c(x_i)$ and order the state space X so that $c_1 \leq c_2 \leq \cdots \leq c_{|X|}$. We remark that in most instances $|X|$ is exponentially large in terms of the size of the problem. In the case of packing n rectangles, for instance, result 1 on page 5 implies that $|X| = (n!)^2$.

The validity in treating annealing as a Markov chain lies in the fact that, given we are in a state x_i, the transition to the next state is independent of what configurations we were in before visiting that state.

From state x_i the algorithm generates, say, x_j as a candidate for the next state. We will denote the probability of accepting this candidate state for the new configuration by

$$a_{ij}(t) = \begin{cases} e^{-(c_j - c_i)/t} & \text{if } i < j \text{ and } t \neq 0, \\ 0 & \text{if } i < j \text{ and } t = 0, \\ 1 & \text{if } i \geq j, \end{cases}$$

where t is the current temperature.

In a given perturbation scheme, not every state is likely to be generated as a candidate state from the current state. (In fact, we will see later why we wouldn't *want* this to be the case.) For $x_i \in X$, the *neighborhood* of x_i, $N(x_i)$, is the set of all states capable of being generated as candidate next-states. If n_{ij} is the probability of generating a candidate state x_j given that we are in a state x_i, we have $N(x_i) = \{x_j : n_{ij} > 0\}$.

If N is the $|X| \times |X|$matrix given by (n_{ij}), we say that N is *irreducible* iff for any two states x_i and x_j, there exists a finite sequence of k states $x_{i_1},\ldots,x_{i_k}$ such that $x_i = x_{i_1}$, $x_j = x_{i_k}$ and $x_{i_{m+1}} \in N(x_{i_m})$ for $m = 1,2,\ldots,k-1$.

We define a transition matrix of probabilities $P(t) = \overbrace{(p_{ij}(t))}^{|X| \times |X|}$, where $p_{ij}(t)$ is the probability of moving from state x_i to state x_j, and t is the temperature. Using the notation given above, we have

$$p_{ij}(t) = \begin{cases} a_{ij}(t) n_{ij} & \text{if } i \neq j \\ 1 - \sum_{k \neq i} a_{ik}(t) n_{ik} & \text{if } i = j. \end{cases}$$

Finally, we note that under very relaxed conditions, any Markov chain will have a unique equilibrium distribution vector

$$\vec{v}(t) = \left(v_1(t),\ldots,v_{|X|}(t)\right), \sum_{i=1}^{|X|} v_i(t) = 1,$$

where $\vec{v}(t)P(t) = \vec{v}(t)$, and $v_i(t)$ is the probability that the chain in in state x_i.

Intuitively, one can think of this vector as representing the relative frequencies of visits to all the states after a large sequence of walks about the state space.

THE PROBABILITY OF CONVERGENCE:

We are now in a position to review a nice result given by Lundy and Mees [8, pp. 116-119]. We let $\vec{e}_i$ denote the row vector with $|X|$ elements having a value of 1 in the i^{th} component and zeros elsewhere. It is easy to see that if state x_i is a local minimum (so that $x_j \in N(x_i) \Rightarrow i < j$), then $\vec{e}_i P(0) = \vec{e}_i$. With very mild restrictions, the following theorem

shows that if vector $\vec{v}(t)$ is the equilibrium vector at temperature t (so that $\vec{v}(t)P(t) = \vec{v}(t)$), we must have

$$\lim_{t \to 0} \vec{v}(t) = \vec{e}_1$$

rather than $\vec{v}(t) \to e_i$ corresponding with some other local minimum state x_i, $i \neq 1$.

THEOREM:

If N is irreducible and symmetric, then for all $t > 0$ the unique equilibrium vector corresponding to $P(t)$ is

$$\vec{v}(t) = v_1(t)\left(1, a_{12}(t), a_{13}(t), \ldots, a_{1|X|}(t)\right).$$

PROOF:

The irreducibility of N guarantees that $v_1(t) \neq 0$ (cf. [11]).

Suppose, for the moment, that for all i, j, (*) $v_i(t)p_{ij}(t) = v_j(t)p_{ji}(t)$. Then, summing both sides over i gives

$$\begin{aligned}
\sum_{i=1}^{|X|} v_i(t)p_{ij}(t) &= \sum_{i=1}^{|X|} v_j(t)p_{ij}(t), \\
\sum_{i=1}^{|X|} v_i(t)p_{ij}(t) &= v_j(t)\sum_{i=1}^{|X|} p_{ij}(t), \text{ so} \\
\left(\vec{v}(t)P(t)\right)_j &= v_j(t) \text{ for all } j.
\end{aligned}$$

To show that (*) above holds, suppose that $1 < i < j$. Then

$$\begin{aligned}
v_j(t)p_{ji}(t) &= \left(v_1(t)a_{1j}(t)\right)\left(a_{ji}(t)n_{ji}\right) \\
&= v_1(t)a_{1j}(t)n_{ji} && \text{(since } i < j \Rightarrow a_{ji}(t) = 1\text{)} \\
&= \left(v_1(t)a_{1i}(t)\right)\left(a_{ij}(t)n_{ji}\right) && \text{(by properties of } e^x\text{)} \\
&= \left(v_1(t)a_{1i}(t)\right)\left(a_{ij}(t)n_{ij}\right) && \text{(by symmetry)} \\
&= v_i(t)p_{ij}(t).
\end{aligned}$$

The proof is virtually the same if $1 < j < i$, and trivial if $i = j$.

Lundy and Mees also point out four interesting conclusions which follow immediately from the form of the equilibrium vector given above:

1. The equilibrium distribution is independent of the topology generated by N (assuming, of course, that we have symmetry).
2. $\vec{v}(t) \to \vec{e}_1$ as $t \to 0$.
3. $\vec{v}(t) \to \left(\frac{1}{|X|}, \ldots, \frac{1}{|X|}\right)$ as $t \to \infty$.
4. At equilibrium, the probability that we are in a state x_i with $c_i - c_1 \leq \varepsilon$ is *at least* $1 - (|X| - 1)e^{-\varepsilon/t}$.

The last item is especially interesting. It states that, if we make the temperature small enough, we can (with probability as close to one as we wish) get as close to the global minimum as we wish. Contrary to claims by some that we must carefully adjust the

temperature schedule in order for annealing to "work," any sufficiently low temperature is highly likely to succeed.

There is, of course, a *faux pas* in the above remark. It is that we have no idea how long convergence will take. In the physical world, annealing could well "work" if we began at low temperatures. We just might have to wait a googolplex of years for the result! The reason we start with a high temperature and gradually lower it is because this approach corresponds with the physical analogy and seems to work well in practice. There are a few papers that discuss methods for varying temperature, [2, 3, 8], but none seem to have any theoretical reasons for deciding why one temperature scheme would be preferable to another. Different temperature schemes do have a profound effect on the performance of the algorithm, however, as we shall see shortly.

TEMPERATURE SCHEDULES:

With certain assumptions, Geman and Geman [2] were able to show that if t_n is the temperature at the n^{th} step, $t_n \to 0$, and $t_n \geq \frac{C}{\log(1+n)}$ for some constant C, the annealing algorithm will converge to the global minimum with probability converging to 1 (as $n \to \infty$). Unfortunately, the value of C they need for applying the result to raster replacement is 20,000. They also claim that reaching a temperature of 0.5 "quickly" and then lowering it according to the above equation worked well in practice. (Note that using the suggested equation from the beginning would require $e^{40,000}$ steps to get the temperature to 0.5.) The fact that Geman and Geman recommended a value of $C = 4$ for practical runs emphasizes the heuristic state that any theory regarding temperature control is currently in.

Hastings [3] claims to have been able to relax some of the restrictions in [2] and gets a similar result with t_n of order $\frac{1}{[\log(n)]^{1-\varepsilon}}$ for $n \geq 2$.

Lundy and Mees [8] argue for an appealing heuristic for varying t. It can be summarized as follows:

1. Choose t_o large enough so that the equilibrium vector $\vec{v}(t)$ is close to uniform. That is, $\vec{v}(t) \approx \left(\frac{1}{|X|},\ldots,\frac{1}{|X|}\right)$. By the form of the equilibrium vector given in the theorem on page 7, this means that $a_{1j}(t_o)$ should be close to 1 for all j. But by the properties of the $a_{ij}(t)$ (see page 7), we must have $e^{-(c_j - c_1)/t_o}$ close to 1 for all j. To ensure this, we should choose $t_o >> c_{|X|} - c_1$. This is not usually difficult to find. In the case of rectangle packing, it could be some constant multiple of the cost obtained by laying the rectangles end-to-end along the longer of their two dimensions.

2. Try to make $\vec{v}(t_{i+1})$ close to $\vec{v}(t_i)$. The idea here is that if we were in equilibrium at step i, we would (hopefully) be close to equilibrium at step i+1. Lundy and Mees derive from this that the temperature schedule should be set at

$$t_{i+1} = \frac{t_i}{1+\beta t_i},$$

where $\beta << \frac{1}{t_o}$.

3. Choose the final temperature t_f in such a way that

$$t_f << \frac{\varepsilon}{(\log|X|-1)-\log\alpha},$$

where ε is the chosen error term of the difference in cost between the final state and the global minimum, and α is the probability for this error as computed from item 4 on page 8.

An easy induction argument gives from the above that $t_n = \frac{t_o}{1+n\beta t_o}$, and hence the schedule will terminate in order $\log|X|-\log\alpha$ time.

The above schedule was used for all cases tested in this paper. In practice, the $\log|X|-\log\alpha$ time bound is still much too long, which is why an arbitrary cut-off variable of 40,000 or 50,000 steps was introduced. In any case, for the examples in this paper, the temperature was usually so low by then that the system appeared to have been "frozen" for quite some time, so that only down-hill steps were being accepted at the point the algorithm terminated. When this is likely to happen, a possible modification to the algorithm would be to stop the process when t reaches a certain "low" value (determined by machine accuracy and problem structure). Conceivably, a list of candidate neighbors could then be systematically searched for lower costs with the process stopping if the end of the list is reached. If a down-hill step were found, a new list could be generated and the process restarted. It seems reasonable to suspect that the length of each list is a polynomial in n, the size of the problem ($n(n-1)/2$ for the traveling salesman), but the number of lists we search might not be. To solve this, we could set up an upper bound on the number of iterations made in each case.

Generally, we must not begin at a temperature that is too low. The reason for this is that at low temperatures, eigenvalues of the transition matrix P are likely to be very close to 1. (Recall that if state x_i is a local minimum, then $\vec{e}_i P(0) = \vec{e}_i$.) Since convergence to the equilibrium distribution is governed by the second largest eigenvalue of $P(t)$ (see [11]), arriving at the equilibrium vector may take exponentially long. Thus, if we begin at a state "near" a local minimum and at a low temperature, we may wind up trapped at that value for an exponentially long time. Figure 8 shows the results of applying annealing to the traveling salesman using a "normal" vs. a "fast" cooling schedule. Figure 9 gives a comparison of each method at various stages of the algorithm. Notice that the fast cooling schedule initially finds lower cost states than the slow one, since it is less likely to accept cost increases. It appears, however, that it also quickly gets trapped in a local minimum, although not enough experimentation was performed to allow any statistical conclusions to be made.

From figure 8, it is apparent that the global minimum in the case of the traveling salesman cannot always be found by inspection. To find out if the algorithm would find an "obvious" global minimum, it was tested on cities located uniformly around the circumference of a circle. Figure 10 gives the initial and final (after 30,000 steps) configurations.

A RESULT REGARDING PERTURBATIONS:

Recall that in the case of the traveling salesman, differing topologies strongly affected the performance of the algorithm. In [7], Lundy and Mees show that not every perturbation scheme is likely to produce good results. Their argument runs as follows:

Let $u_k(t)$ be the expected number of steps that the algorithm (beginning at state x_k and at temperature t) takes to make its first visit to state x_1. Then

$$u_k(t) = 1 + \sum_{j=2}^{|X|} p_{kj}(t) u_j(t).$$

From this it follows that

$$1 = u_i(t) n_{i1} + \sum_{j=2}^{i-1} n_{ij}\big(u_i(t) - u_j(t)\big) + \sum_{j=i+1}^{|X|} n_{ij} a_{ij}(t)\big(u_i(t) - u_j(t)\big),$$

where the first sum is omitted when $i = 2$ and the second when $i = |X|$. Suppose, now, we have a perturbation scheme in which all states are equally likely to be selected as candidate next-states from a given state. Then the above equation reduces to

$$|X| = u_i(t) + \sum_{j=2}^{i-1}\big(u_i(t) - u_j(t)\big) + \sum_{j=i+1}^{|X|} a_{ij}(t)\big(u_i(t) - u_j(t)\big),$$

and for all $a_{ij}(t)$, the solution is $u_i(t) = |X|$ for all $i \geq 2$. Hence, $\frac{1}{|X|}\sum_{k=2}^{|X|} u_k(t) = |X| - 1$. That is, if the initial state is chosen randomly, the expected time to first visit the global minimum is the size of the state space itself.

ANNEALING VS. DETERMINISTIC DESCENT:

Throughout many of their stages, computational costs for annealing and deterministic descent techniques are the same. Both must perturb the problem structure, evaluate the cost of the new configuration, and compare the new cost with that of the old. With annealing there is an additional resource drain in evaluating an exponential and computing new temperatures. Because of this, it might be argued that rather than consuming resources in this manner, one should use that computation time instead to conduct repeated applications of some deterministic descent method.

If a problem is not likely to have many local minima, this may be a valid point. Making that determination, however, would probably not be easy. In addition, Lundy and Mees give an example in [7] and again in [8] showing there are cases where annealing provably performs much better than a deterministic descent algorithm run several times. The example is perhaps somewhat contrived, but it is worth reviewing:

Let $V = [-N, N] \times [-N, N] \subset R^2$, where N is a large integer. Let X be the the discrete set of points obtained by intersecting V with a "grid of points" of mesh size δ as shown in figure 11. Define the cost function $c: X \to R$ by

$$c(x_1, x_2) = \begin{cases} l_\infty\big(|x_1|, |x_2|\big) & \text{if } l_\infty\big(|x_1|, |x_2|\big) \neq n + \delta,\ n \in Z^+ \\ n - \varepsilon & \text{otherwise.} \end{cases}$$

Suppose, now, that $\varepsilon << t << \delta$. With annealing, any cost increases will be of size $s = \delta, \varepsilon$, or $\varepsilon + 2\delta$. The choice of t assures us that $e^{-s/t}$ is arbitrarily close to zero except when $s = \varepsilon$ in which case it is close to one. Thus, with probability as close to one as we

wish, the annealing method will march to the global minimum (which clearly occurs only at (0,0)) in O(N) time.

The deterministic descent method, however, will reach the global minimum *only* if it is begun inside $[-1,1]\times[-1,1]$, so its expected time to arrive there is of order N^2. Furthermore, if we were to extend the example to R^k, the time for annealing would remain at order N vs. order N^k for deterministic descent.

CONCLUDING REMARKS:

There are at least two recent books in print which discuss simulated annealing in some detail [2, 6]. Given its impressive results, this is not too surprising. It is an easy algorithm to use, provided a reasonable perturbation method can be found. The rectangle packing example shows, however, that finding a nice swapping scheme may not be an entirely trivial matter. Although simulated annealing is being used in an ever increasing array of applications, real progress regarding the *theory* of the algorithm is somewhat bogged down. Not much is known regarding the *speed* of convergence of the algorithm. The known results tend to be largely negative, or require unbelievable constraining assumptions [7, 8]. Part of the problem lies in the difficulty in getting a handle on how convergence is affected by the topology of the problem structure, which in turn is determined by the perturbation method chosen. Any results which could be found here would greatly help in the understanding of this remarkable algorithm.

REFERENCES:

[1] Aarts, E., and Korst, J. *Simulated Annealing and Boltzman Machines*, Wiley, 1989.

[2] Geman, S., and Geman, D., Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images, IEEE *Transactions on Pattern Analysis and Machine Intelligence*. **6**, (November 1984), pp. 721-741.

[3] Hastings, H.M., Convergence of Simulated Annealing, Hofstra University Report (1985).

[4] Complementary Partial Orders and Rectangle Packing, University of Edinburgh Internal Report CSR-190-85.

[5] Kirkpatrick. S., Gelatt, C., and Vecchi, M. Optimisation by Simulated Annealing, *Science* **220** (May 1983), pp. 671-680.

[6] Laarhoven, P.J.M. *Simulated Annealing: Theory and Application*, D. Reidel, 1989.

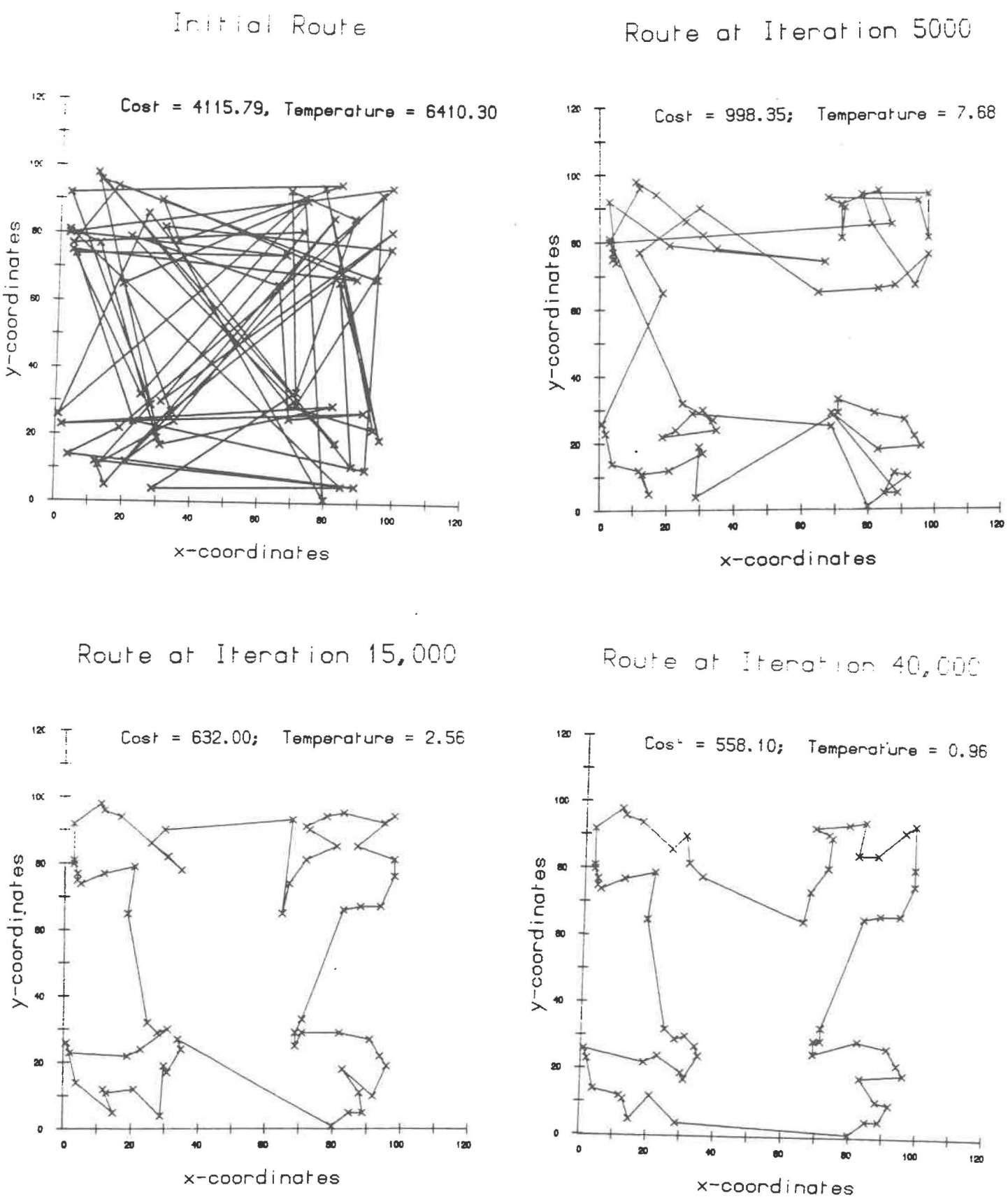
[7] Lundy, M., and Mees, A., Convergence of the Annealing Algorithm, Cambridge University Report (1983).

[8] Lundy, M., and Mees, A., Convergence of An Annealing Algorithm, *Mathematical Programming* **34** (1986), pp. 111-124.

[9] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., and Teller, A.W., Equation of State Calculation by Fast Computing Machines., *Journal of Chemical Physics*, **21** (1953) pp. 1087-1092.

[10] Nahar, S., Sahni, S., and Shargowitz, E., Experiments with Simulated Annealing, *Proceedings*, 22nd Design Automation Conference (1985) pp. 748-752.

[11] Seneta, E. *Non-negative Matrices and Markov Chains*, Springer-Verlag, 1981.



Figures 1a-d: Annealing Applied to the Traveling Salesman.

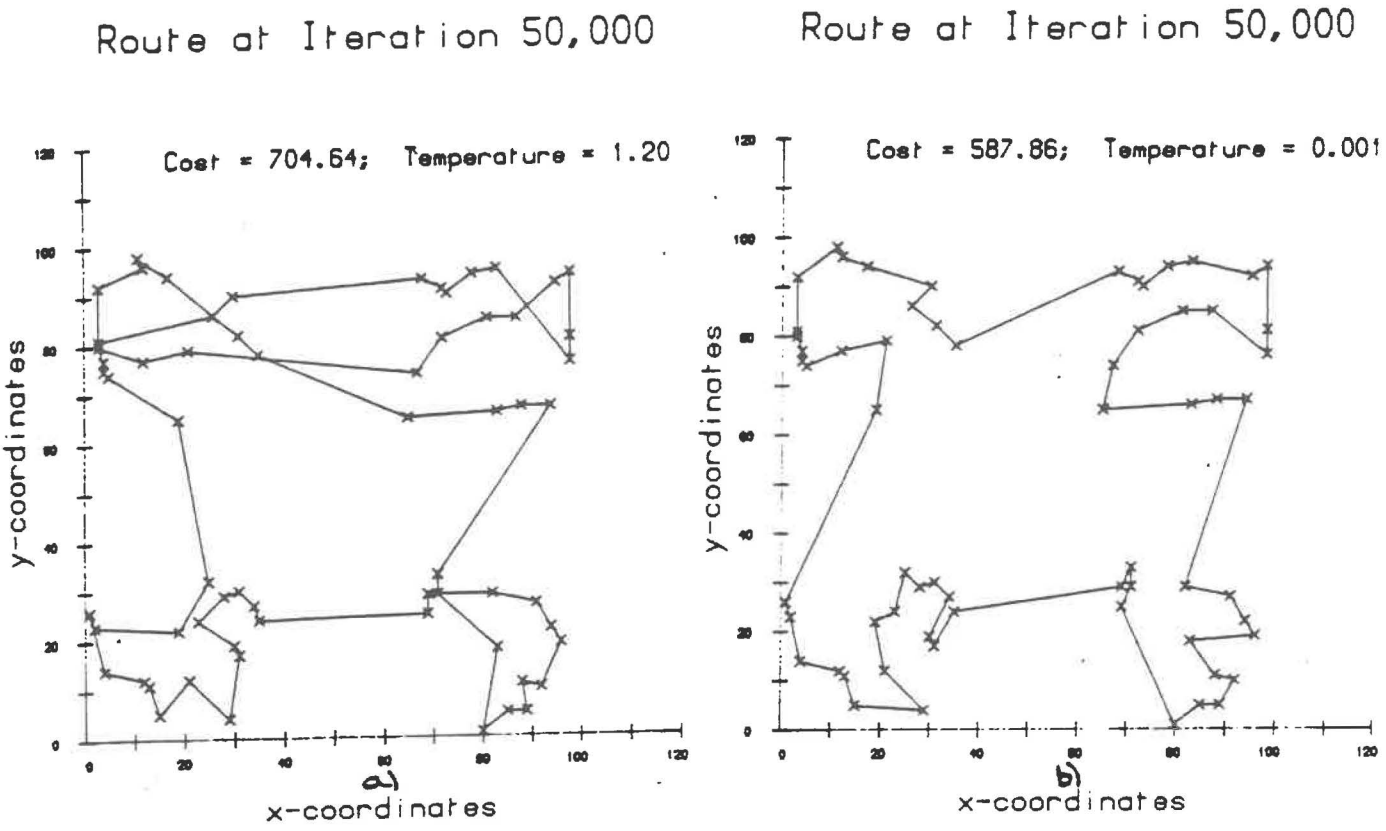


Figure 2: Comparison of Perturbation Schemes. Pair Swap (a); Route Reversal (b).

Pair Swap Average ($\overline{X_1}$)	Route Reversal Average ($\overline{X_2}$)	t-value
796.9	566.0	< 0.01

Figure 3: A t-test for H_o: $\mu_1 \leq \mu_2$ vs. H_a: $\mu_1 > \mu_2$. μ_1 and μ_2 represent the population mean final costs for pair swapping vs. route reversal.

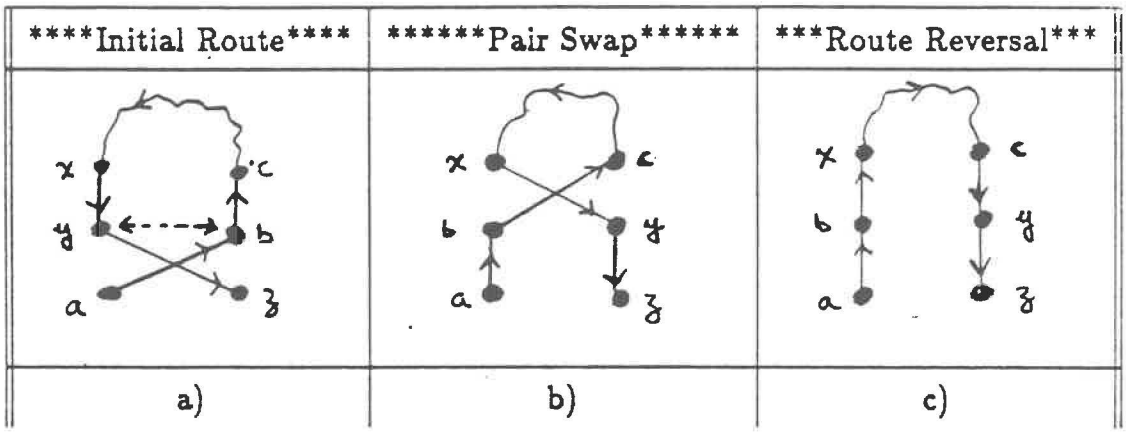


Figure 4: Initial Route (a); Route After Pair Swapping (b); Route After Pair Swapping with Intervening Route Reversal (c).

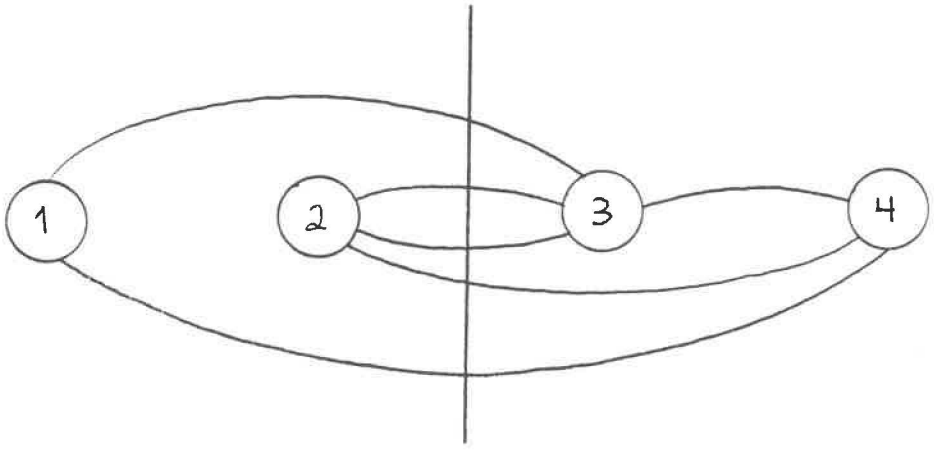


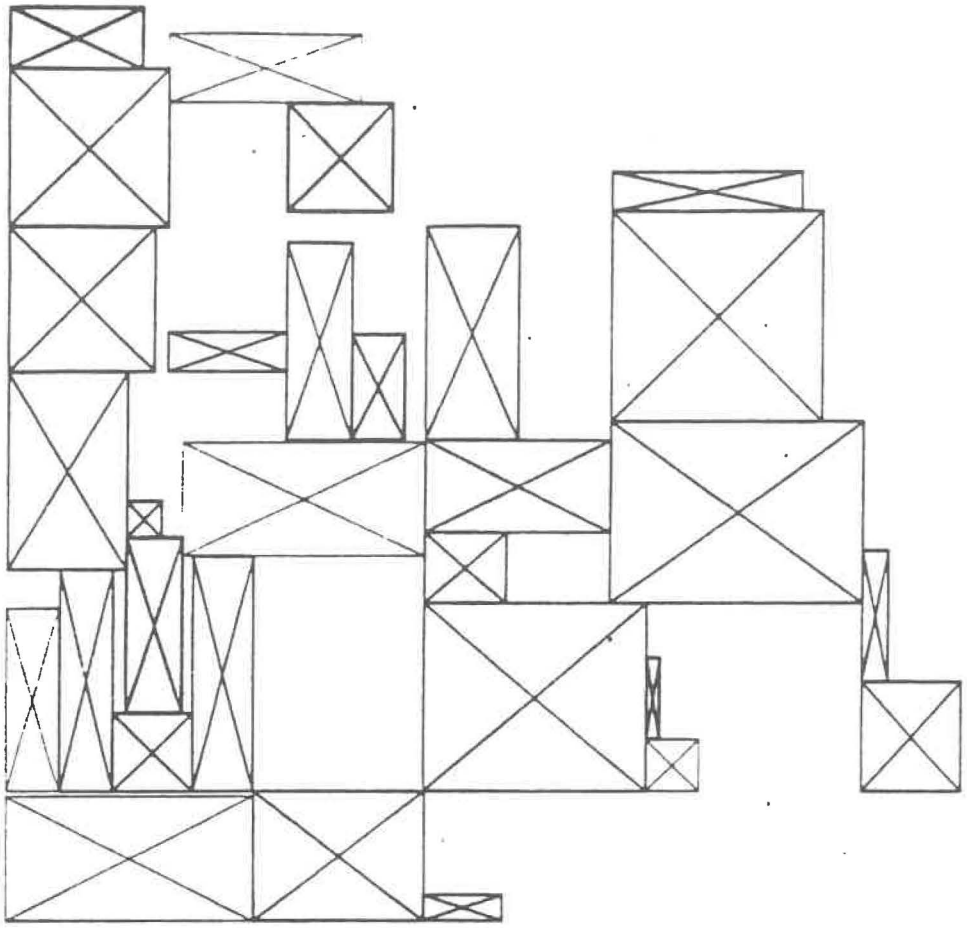
Figure 5: A GOLA Instance with Density 5.

Iteration Number	Best Cost Using Nahar's Method	Best Cost Using the Standard Annealing Method
0	39	39
10,000	13	8
20,000	10	6
30,000	10	5
40,000	9	5
50,000	9	5

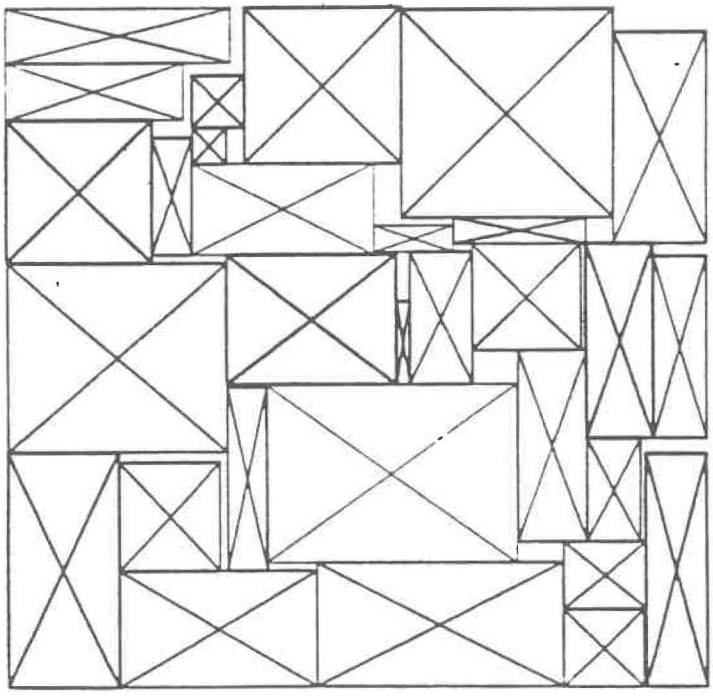
Figure 6a--Nahar's Method vs. Annealing for GOLA.

Iteration Number	Best Cost Using Nahar's Method	Best Cost Using the Standard Annealing Method
0	4115.80	4115.80
10,000	1528.92	776.18
20,000	1528.92	658.16
30,000	1528.92	586.89
40,000	1528.92	572.77
50,000	1528.92	562.36

Figure 6b--Nahar's Method vs. Annealing for the Traveling Salesman.



a)



b)

Figure 7: Annealing Applied to Rectangle Packing After 1000 Iterations (a); After 50,000 Iterations (b).

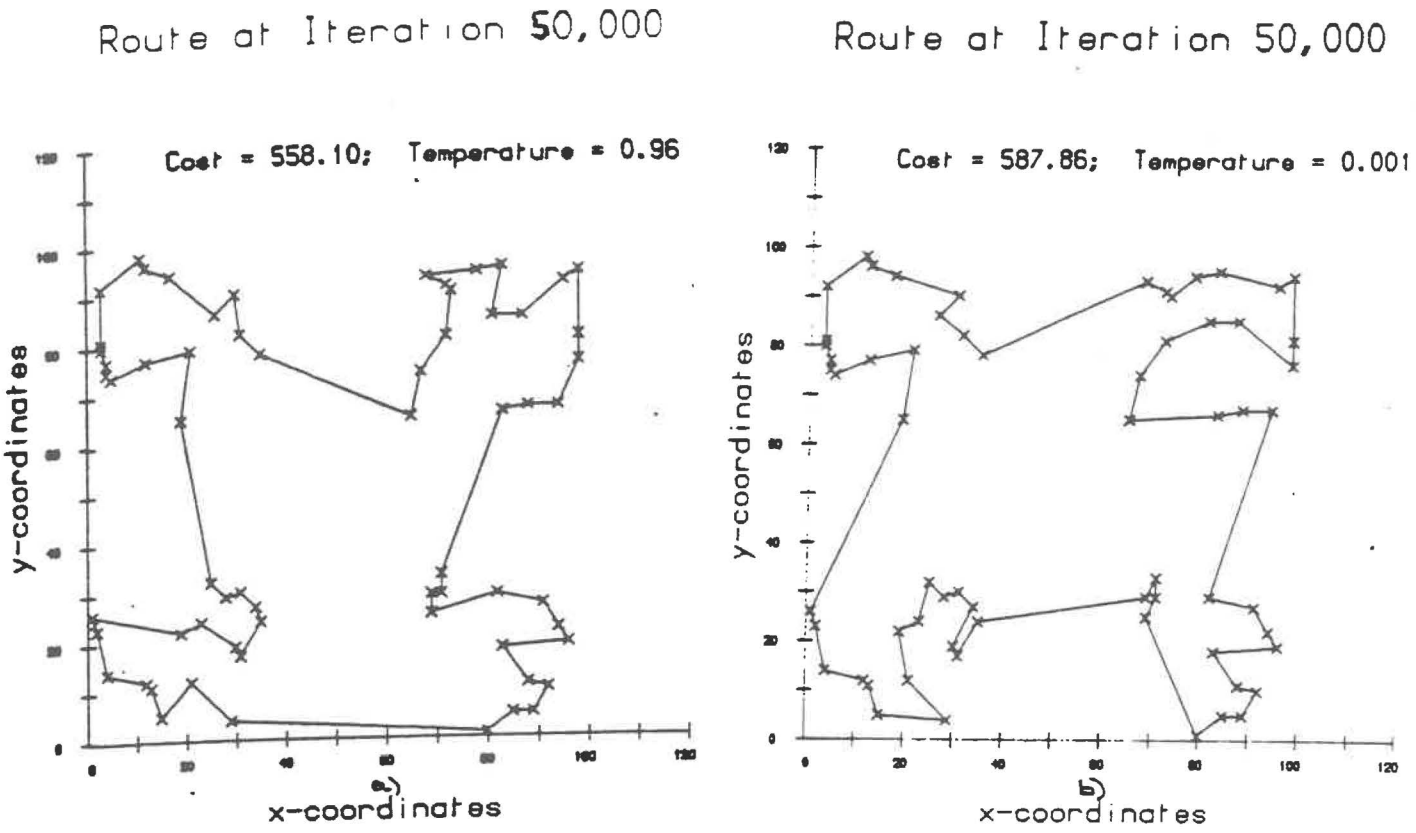


Figure 8: Standard (a) vs. "Fast" (b) Cooling Schedules.

Iteration Number	Best Cost Using Lundy and Mees' Cooling Schedule	Best Cost Using a "Fast" Cooling Schedule
0	4115.80	4115.80
10,000	776.18	593.38
20,000	658.16	587.86
30,000	586.89	587.86
40,000	572.77	587.86
50,000	558.10	587.86

Figure 9--Standard vs. "Fast" Cooling Schedules.

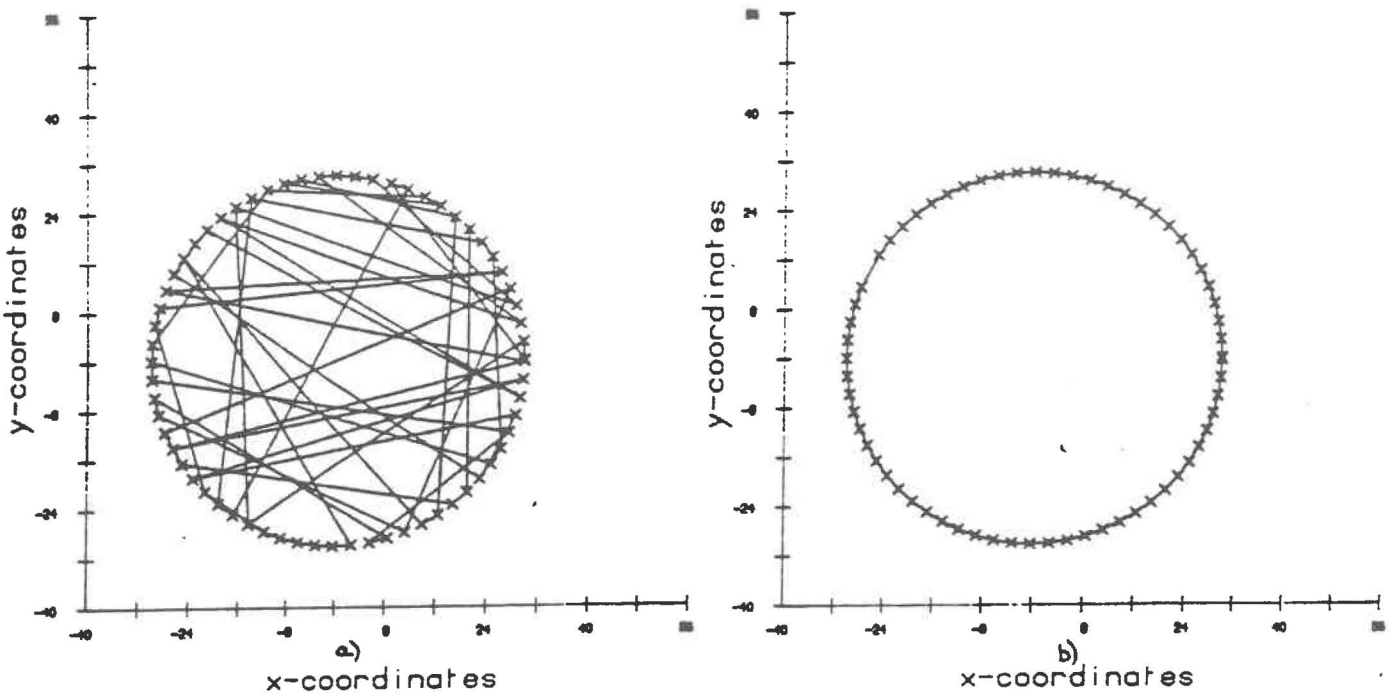


Figure 10: Annealing Finding the Global Minimum in 30,000 Steps.

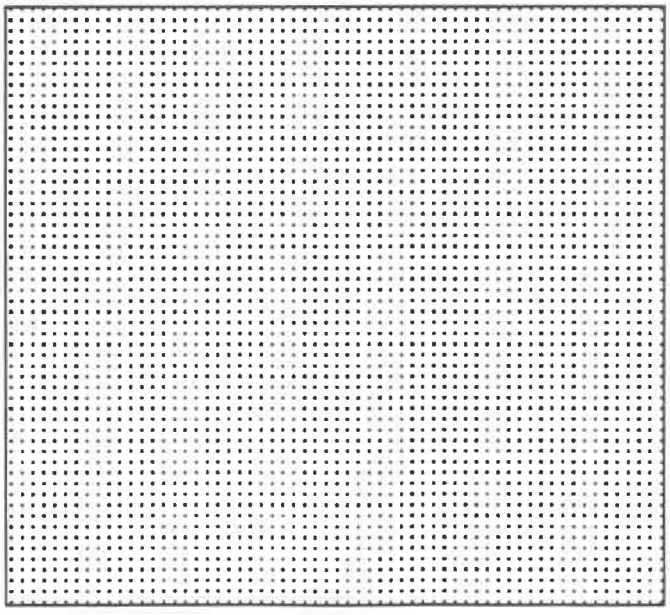


Figure 11: A Structure on which Annealing Provably Performs Better than Deterministic Descent.